N73-11185

NASA CONTRACTOR
REPORT

NASA CR-128996

MEMORY INTERFACE SIMULATOR: A COMPUTER DESIGN AID

By D. S. Taylor, T. Williams,
and J. E. Weatherbee
Computer Sciences Corporation
Field Service Division
Aerospace Systems Center
8300 S. Whitesburg Drive
Huntsville, Alabama 35802

October 6, 1972

CASE FILE
COPY

Prepared for

NASA-GEORGE C. MARSHALL SPACE FLIGHT CENTER
Marshall Space Flight Center, Alabama 35812

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.
NASA CR-128996

2. GOVERNMENT ACCESSION NO.

3. RECIPIENT'S CATALOG NO.

4. TITLE AND SUBTITLE
Memory Interface Simulator: A Computer Design Aid

5. REPORT DATE
6 October 1972

6. PERFORMING ORGANIZATION CODE

7. AUTHOR(S)
Dr. D.S. Taylor, Dr. T. Williams, and Dr. J.E. Weatherbee

8. PERFORMING ORGANIZATION REPORT #

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Computer Sciences Corporation
Field Services Division, Aerospace Systems Center
8300 S. Whitesburg Drive
Huntsville, Alabama 35802

10. WORK UNIT NO.

11. CONTRACT OR GRANT NO.
NAS8-21805

12. SPONSORING AGENCY NAME AND ADDRESS
National Aeronautics and Space Administration
Washington, D.C. 20546

13. TYPE OF REPORT & PERIOD COVERED
Contractor Report

14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

16. ABSTRACT

This report presents the results of a study conducted with a digital simulation model being used in the design of the Automatically Reconfigurable Modular Multiprocessor System (ARMMS), a candidate computer system for future manned and unmanned space missions. The model simulates the activity involved as instructions are fetched from random access memory (RAM) for execution in one of the system central processing units (CPUs). Microprogrammed CPU's were assumed for this study and a series of model runs were made which measured instruction execution time under various assumptions pertaining to the CPU's and the interface between the CPU's and RAM. Design tradeoffs are presented in the following areas:

1) bus widths

2) CPU microprogram read only memory cycle time

3) multiple instruction fetch

4) instruction mix

17. KEY WORDS
Spaceborne Computer
Central Processing Unit
Configuration
Interface
Microprogram
Random Access Memory
Read Only Memory
Simulation

18. DISTRIBUTION STATEMENT
Bobby Hodges
Mr. B. Hodges
Computer Systems & Simulation Division
MSFC/Computation Laboratory
Unclassified-Unlimited

19. SECURITY CLASSIF. (of this report)
Unclassified

20. SECURITY CLASSIF. (of this page)
Unclassified

21. NO. OF PAGES
24

22. PRICE
NTIS

TABLE OF CONTENTS

ACKNOWLEDGEMENT

The authors wish to acknowledge the project monitor, Mr. B. Hodges (S&E-COMP-S) for technical contributions and liaison work with Astrionics Laboratory and also his helpful discussions throughout the duration of this project.

Thanks is also due to Dr. J.B. White (S&E-ASTR-CA) for his continued cooperation and encouragement during the course of this work.

SUMMARY

The Astrionics Laboratory of the Marshall Space Flight Center, Huntsville, Alabama, is currently designing a spaceborne computer system, the Automatically Reconfigurable Modular Multiprocessor System (ARMMS). This report presents the results of a study conducted with a digital simulation model being used in the ARMMS design. The model simulates the activity involved as instructions are fetched from random access memory (RAM) for execution in one of the system central processing units (CPU's).

The time required for the execution of an instruction is a function not only of the internal speed of the CPU and the memory cycle time but also the design of the Memory-Processor interface and the amount of interference produced at this interface when more than one CPU attempts to access the same memory bank in RAM. Simulation of the instruction execution activity allows all of these factors to be considered while measuring the effective execution time under various design assumptions.

In this study the basic ARMMS configuration was assumed to consist of a number of microprogrammable CPU's connected to a number of banks of RAM by two sets of one-way busses; one bus set used to transmit memory addresses from the CPU's to RAM and the other used to return instructions and data from RAM to the CPU's. Design tradeoffs are presented in the following areas:

1) bus widths

2) CPU microprogram read only memory cycle time

3) multiple instruction fetch

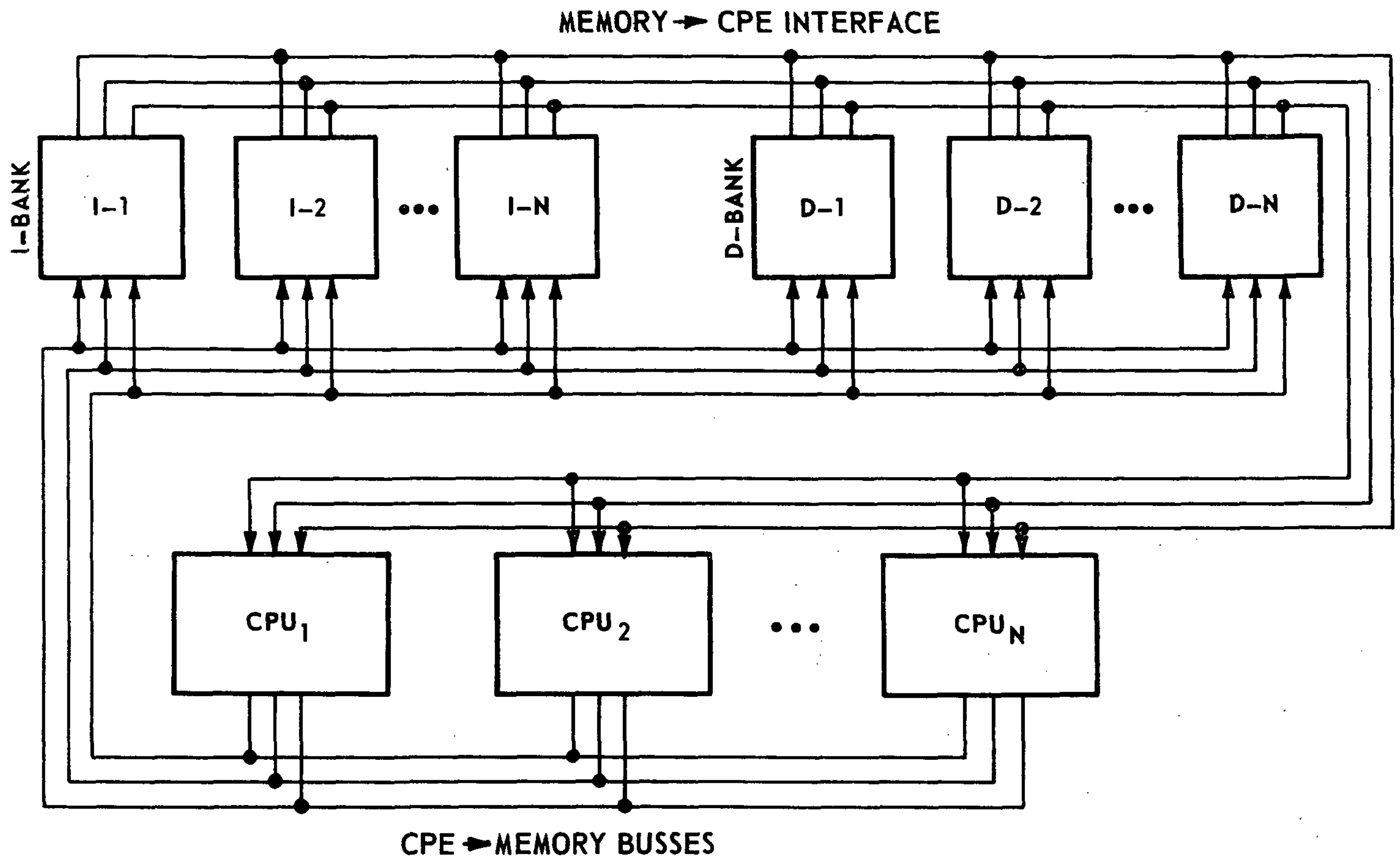
4) instruction mix

SECTION I. INTRODUCTION

As the duration and complexity of space missions increase, the requirements placed on the onboard digital computing equipment also increase. Future missions, such as the earth-orbiting space station and astronomical space observatory will be measured in years instead of days. Onboard computing tasks are being expanded to include resource management and experimental data processing. These requirements make it imperative that spaceborne computers in the late 1970's and 1980's be characterized by both high reliability and high computing capacity. The Astrionics Laboratory of the Marshall Space Flight Center, Huntsville, Alabama, is currently designing a spaceborne computer system, the Automatically Reconfigurable Modular Multiprocessor System (ARMMS), which can satisfy both of these requirements [1].

In a previous report [2], the use of digital simulation in the ARMMS design was demonstrated through an exercise in which two simulation models were used to obtain an optimal ARMMS configuration for a hypothetical mission. The purpose of this report is to present the results of a study conducted with an updated version of one of these models, the Memory-CPE Interface Model.

Historically, the data processing workloads for spaceborne digital computer systems have been characterized as primarily computation so that the speed and efficiency with which instructions are fetched from memory and executed are critical to overall system performance. The ARMMS Memory -CPE Interface Simulation Model was developed to study, through simulation, the effect of various design concepts on the speed of execution of instructions in the ARMMS. Figure 1 shows the general processor memory configuration simulated in this study. Two sets of one-way busses interconnect the Central Processing Element (CPE), composed of a number of Central Processing Units (CPU's), with a number of banks of Random Access Memory (RAM). One bus group is used to transmit memory addresses from the CPE to RAM and the other is used to return instructions and data from RAM to the CPE.



MEMORY-CPE INTERFACE

Figure 1

SECTION II. SYSTEM CONFIGURATION ASSUMPTIONS

A. Memory-CPE Interface

The memory-CPE interface consists of three fundamental pieces of hardware

(1) central processing units (CPUs)
(2) memory modules
(3) busses

Each of these items affects the system operation, not only in terms of their numbers, and speed, but also in their operational concept. For example, the memory modules may be logically separated into instruction (I) and data (D) banks. Also, the CPUs may or may not time share the busses.

In order to perform an instruction level simulation of this interface, assumptions need to be made in the following areas:

(1) memory operation/speed
(2) bus operation/speed
(3) CPU operation/speed

The memory operation can be simulated by selecting a memory read cycle time and memory read access time. The operation of the busses is also easily simulated by specifying a bus width (in bits) and the time required to transmit one bit stream across the bus. However, the operation of a CPU is more complicated and an understanding of how the CPU fetches and executes individual instruction is necessary for proper simulation of this activity.

B. Central Processing Units

A candidate processor for ARMMS is the Space Ultrareliable Modular Computer (SUMC) [5], a microprogrammable processor being developed at MSFC. The processors assumed in this study are like SUMC in that they are also microprogrammable and have internal logic which is being considered for SUMC. This microprogramming allows the construction of a large number of unique aerospace instructions from a much smaller number of microinstructions. It is assumed that Microprogram Read Only Memory (MROM) in the CPU contains the prestored sequences of internal microinstructions required to fetch and execute the program instructions. The processors are also assumed to contain a small buffer memory which may be used for temporary data storage by a programmer or for instruction retention in the CPU by the system executive. Traffic across the memory-CPE interface is generated during the fetch cycle of instruction executions in the CPU's.

C. Fetch Cycle

A flow chart of the fetch cycle is presented in Figure 2. Every instruction requires a fetch cycle since a fetch is executed to read program instructions from memory for subsequent execution.

At most six distinct steps are involved in the fetch cycle with each step requiring at least one MROM cycle.

1. The program counter (PC) is incremented by one and stored in the memory address register (MAR). Main memory control is set for a read and a memory read cycle is initiated if the instruction is not in the CPU buffer memory.

2. This step involves looping through one or more MROM accesses; the exact number depends on the location of the instruction to be executed. If the instruction is in the CPU buffer memory it is moved to the memory register (MR) in one MROM cycle time. If the instruction must be fetched from main memory the CPU clock advances an integer multiple of MROM cycles until the instruction has been moved from main memory to the MR.

3. The instruction is moved from the memory register to the instruction register (IR). The address displacement field (MRD) is summed with the contents of the index register (X) specified by the instruction index field and the result is placed in the Product Remainder Register (PRR).

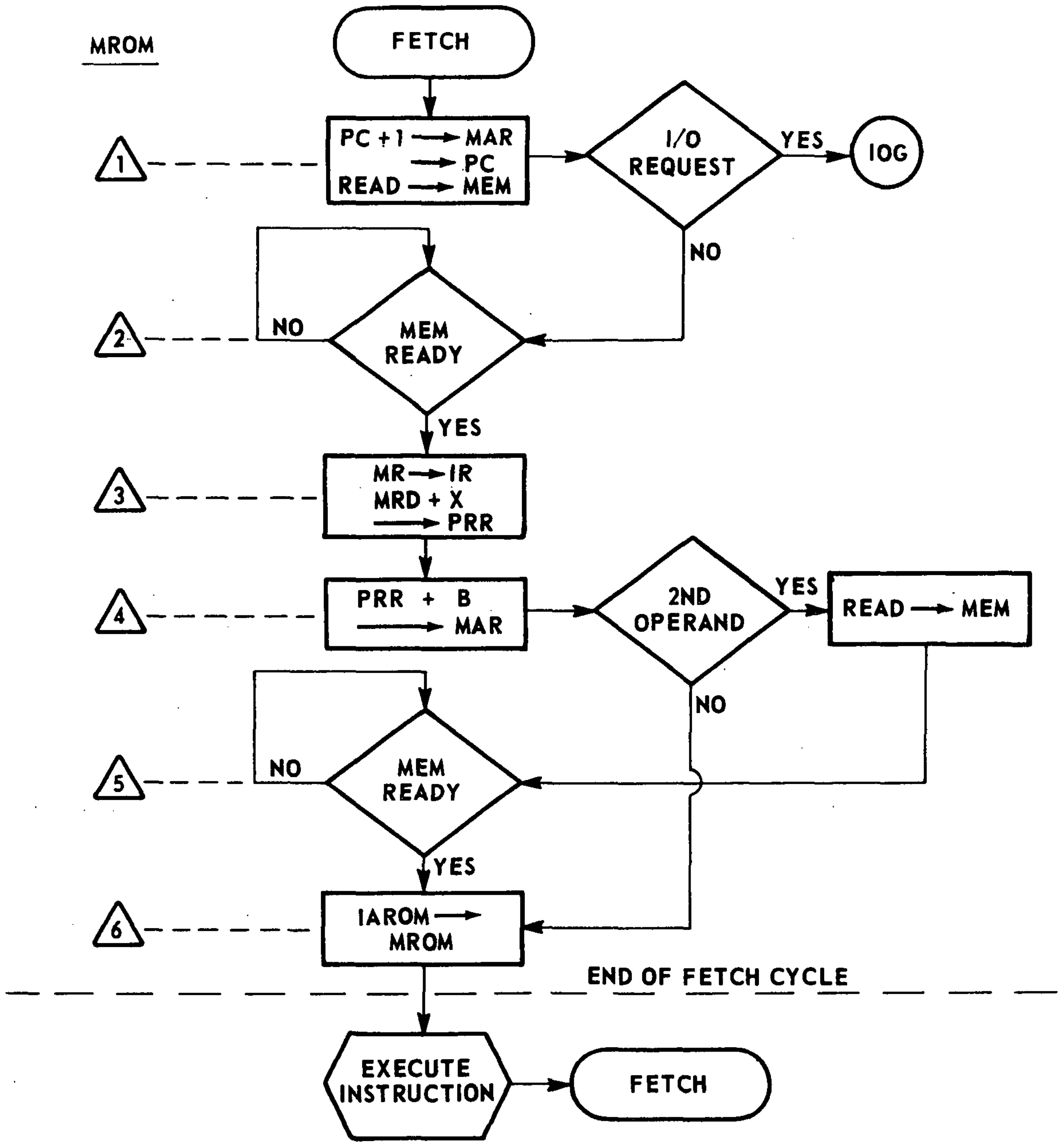
4. The content of the PRR is summed with the content of the base register (B) specified by the instruction base field and the result is placed in the memory address register. If a second operand is required main memory control is set for a read and step 5 is performed; otherwise step 6 is performed next.

5. The CPU clock advances an integer multiple of MROM cycles until the operand specified by the content of the MAR has been moved to the MR.

6. The starting address, in MROM, for the microinstructions required to execute the specified instruction is fetched from the instruction address read only memory (IAROM).

At the end of step 6 the desired instruction is ready for execution in the CPU. The time required to complete the execution cycle depends of course on the type of instruction being executed. From Figure 2 it is readily observed that the time required to complete the fetch cycle is not the same for all instructions. An analysis of the various routes that a fetch cycle may encounter is shown in Table 1.

The action of the memory-CPE interface simulator in executing the fetch cycle is illustrated in Figure 3. The MROM numbers on the simulated fetch cycle (Figure 3) are presented in order to compare the simulated fetch cycle with the actual fetch cycle (Figure 2).



FETCH CYCLE

FIGURE 2

TABLE 1. FETCH CYCLE ROUTES

ROUTE	INSTRUCTION	DATA	TIME REQUIRED FOR FETCH CYCLE
A	NO MEMORY WAIT	NO SECOND OPERAND	5 MICRO INSTRUCTIONS
B	NO MEMORY WAIT	REQUIRES SECOND OPERAND	6 MICRO INSTRUCTIONS
C	REQUIRES MEMORY WAIT	NO SECOND OPERAND	4 + n MICRO INSTRUCTIONS WHERE n = NUMBER OF MICRO INSTRUCTIONS WAITING ON INSTRUCTION
D	REQUIRES MEMORY WAIT	REQUIRES SECOND OPERAND	4 + n + m MICRO INSTRUCTIONS WHERE m = NUMBER OF MICRO INSTRUCTIONS WAITING ON SECOND OPERAND

The most difficult area of the fetch cycle to simulate is the "Memory Ready" loop (MROM 2 and 6). This portion of the fetch cycle is concerned with a CPU sending an address over a bus to memory, obtaining the instruction or data from memory, and then transmitting it back to the CPU. If the MROM clock is not stopped, then the time required to complete this loop must be an integral number of MROM cycles, as depicted in Table 1. A brief discussion of the simulation model is presented in the following section.

SECTION III. MEMORY INTERFACE SIMULATION MODEL

A. General

The Memory Interface Simulator operates under the assumption that there is always a non-empty set of instructions awaiting execution by each CPU. This corresponds to a 100 percent CPU utilization and represents a worst case condition from the standpoint of memory contention. Instructions are classified as either long or short, with the difference being the actual instruction execution time (independent of the fetch cycle time). The selection of either a long or short instruction for execution is determined via a probability distribution derived from a Gibson instruction mix.

A single address instruction format is assumed, together with a bank of General Registers: This means that, in general, one operand is fetched from memory and the other from a General Register.

As Figure 3 illustrates, an instruction fetch and data fetch are not required for all instructions executed. For example, an instruction fetch is not always necessary if a multiple instruction fetch is incorporated in the simulated CPU; similarly, a data fetch is not required for every instruction executed, e.g. JUMP. Values of functions defining the fetch/no fetch ratios are supplied as inputs to the simulator. This fetch/no fetch ratio for instructions is a function of the system being simulated while the data fetch/no fetch ratio is a function of the instruction mix.

The output from the Memory Interface Simulator is the average instruction execution time for both long and short instructions. This average instruction execution time is computed after simulating the execution of a large number of instructions (currently 5000), and consists of the following incremental times

(1) Instruction address transfer time

(2) Instruction memory access time (if required)

(3) Instruction transfer time

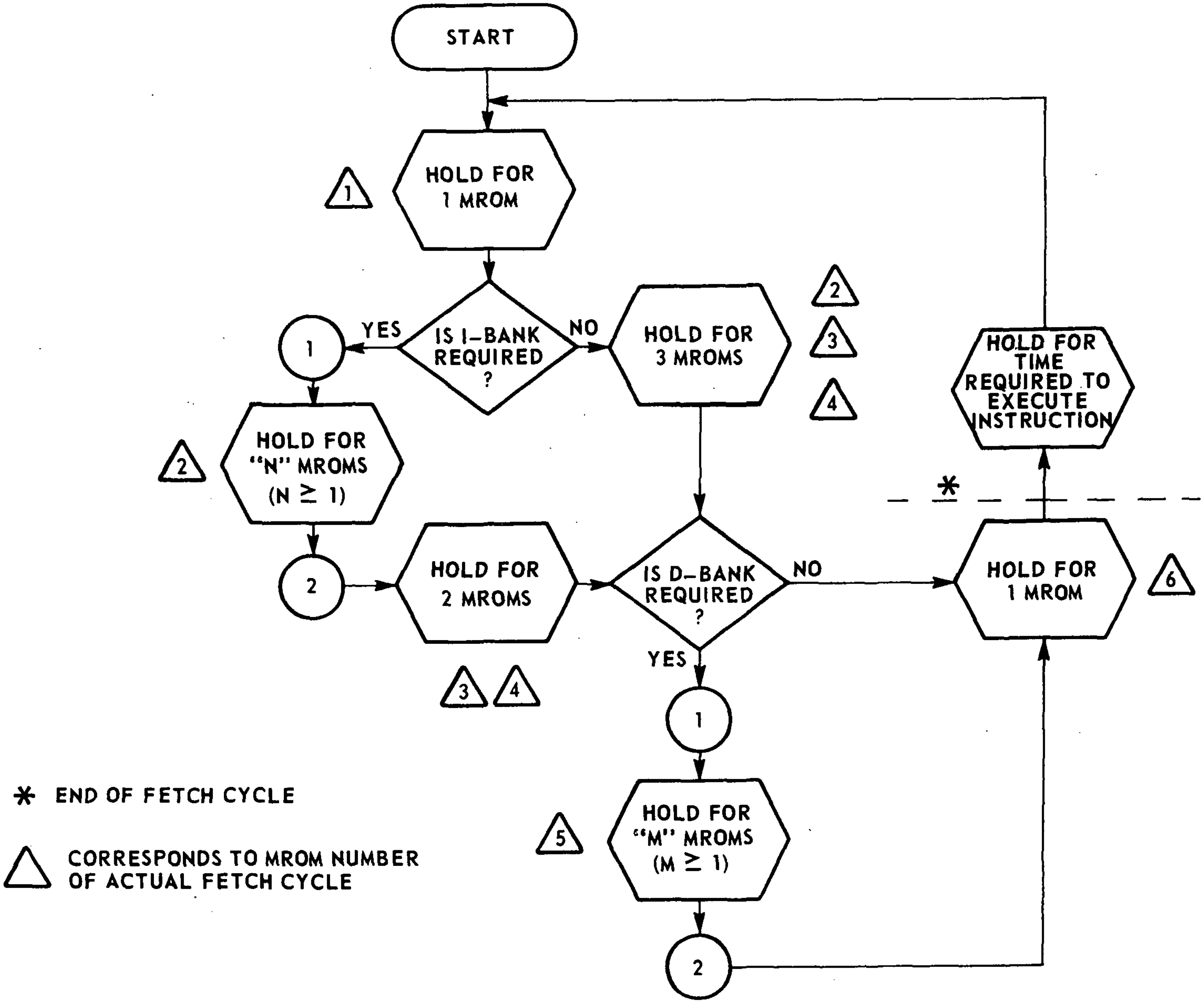
(4) Data address transfer time

(5) Data memory access time (if required)

(6) Data transfer time

(7) Any interference or idle time (queue)

(8) CPU instruction execution time.



SIMULATED FETCH CYCLE

FIGURE 3

B。 Baseline Interface Parameters

The "baseline configuration" of the memory-CPE Interface Simulator is depicted in Table 2。 All future parameter changes will be compared to this baseline. Since it is not possible to present the effects of varying all of these parameters, tradeoffs will only be presented for those parameters which are starred in Table 2. The baseline configuration is not meant to be interpreted as the baseline ARMMS configuration. Instead, it merely represents a reference point to which other system configurations can be compared and their relative merit assessed。

TABLE 2. BASELINE CONFIGURATION

PARAMETER	ASSIGNED VALUE
NO. OF CPUs	4
NO. I-BANK MEMORY MODULES	8
NO. D-BANK MEMORY MODULES	16
CPU-MEMORY WORD SIZE	30 BITS
*ALL BUS WIDTHS	15 BITS
BUS SPEED	65 NSEC
MEMORY CYCLE TIME	520 NSEC
MEMORY ACCESS TIME	260 NSEC
*NO. OF INSTRUCTIONS FETCHED PER I-BANK ACCESS	1
MEMORY-CPU WORD SIZE	38 BITS
NO. OF BUSSES	4
*MROM EXECUTION TIME	325 NSEC
*RATIO OF LONG TO SHORT INSTRUCTIONS	1:7
NO. OF INSTRUCTIONS RETAINED IN CPU	8
% INSTRUCTIONS REQUIRING NO SECOND OPERAND	16%
*% DATA RETENTION	0%
MROMs FOR SHORT INSTRUCTION EXECUTION	1
*MROMs FOR LONG INSTRUCTION EXECUTION	11

SECTION IV. CONFIGURATION TRADEOFFS

A. Bus Widths

Figure 4 shows the results of a bus width tradeoff. Two different MROM cycle times were utilized in this tradeoff to emphasize the relationship between bus widths and MROM cycle time. For example, with a 325 nsec MROM cycle time, a 15-bit bus is preferred while a 390 nsec MROM cycle time indicates that a 10-bit bus would be preferable. The discontinuities in the curves are caused by the fact that the fetch cycle requires an integral number of MROMs to be executed for each step in the fetch cycle (reference Figure 2). The difference between the "with queue" and "no queue" curves reflects time lost due to memory conflicts and waiting for the MROM clock to complete a cycle.

This figure, as well as all of the remaining figures, has the baseline configuration clearly indicated. This enables one to readily discern variations in the average instruction execution time for different system configurations.

B. MROM Execution Time

The effect of the MROM execution time on the average instruction execution time is illustrated in Figure 5. Intuitively, one expects the average instruction execution time to increase as the MROM execution time increases. However, Figure 5 shows that this is not necessarily true, as evidenced by the fact that as the MROM execution time increases from 520 to 585 nsec, the average instruction execution time decreases. The reason for this decrease is due to the characteristics of the fetch cycle. Since the "memory ready" portion of the fetch cycle requires an integral number of MROM cycle time advances (reference Table 1), extending the MROM cycle time could result in fewer such cycles being executed during a fetch. Thus, even though the MROM cycle time increases, the fact that there might be fewer MROM cycles, could result in a net reduction of the average instruction execution time. For the baseline parameters chosen, Table 2, this condition occurs for a MROM cycle time between 520 and 585 nsec. In this particular baseline configuration, however, it does not appear that one would take advantage of this reduction, since the average instruction execution time at this point is much higher than the baseline.

C. Multiple Instruction Fetch

Experience has indicated that approximately 85 percent of the time, the next instruction required for execution will be the next sequential instruction in memory. With this in mind, one would like to explore the possibilities of multiple word instruction fetches per I-bank memory access. Figure 6 shows the result of such a tradeoff for this baseline

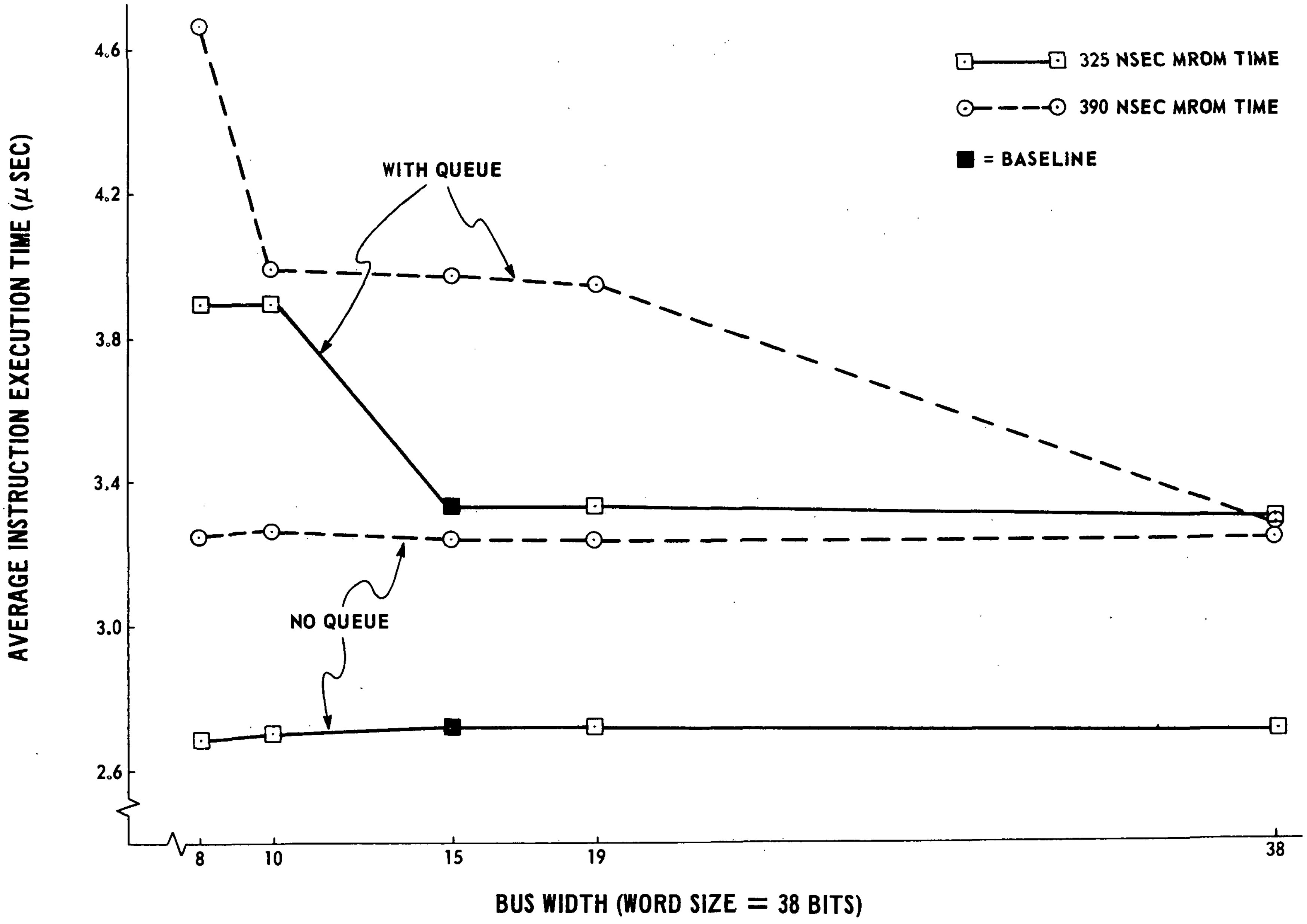

AVERAGE INSTRUCTION EXECUTION TIME (μ SEC)
4.6
4.2
3.8
3.4
3.0
2.6
8
10
15
19
38
BUS WIDTH (WORD SIZE = 38 BITS)
325 NSEC MROM TIME
390 NSEC MROM TIME
■ = BASELINE
WITH QUEUE
NO QUEUE

FIGURE 4

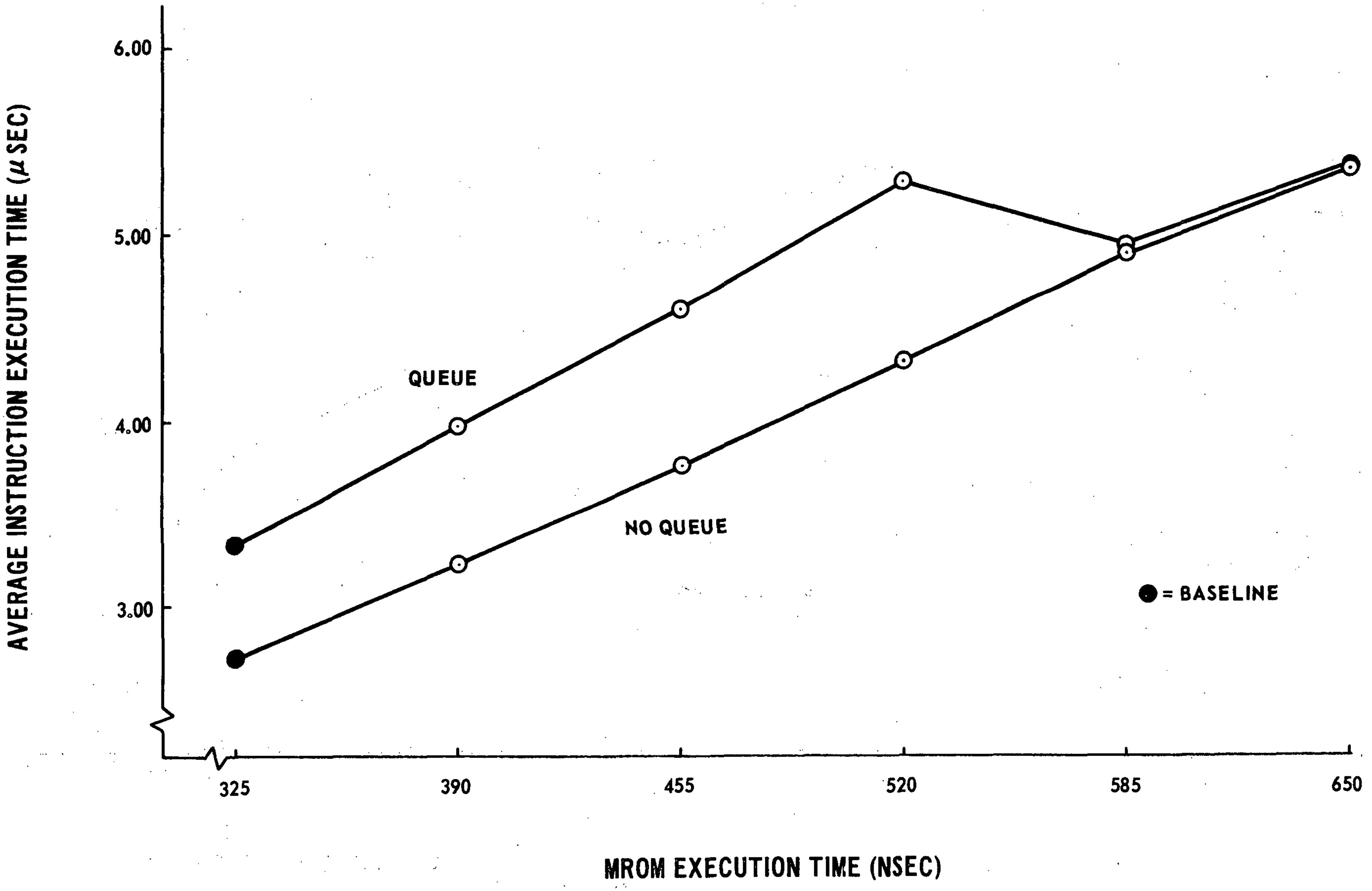

AVERAGE INSTRUCTION EXECUTION TIME (μ SEC)
6.00
5.00
4.00
3.00
QUEUE
NO QUEUE
● = BASELINE
325
390
455
520
585
650
MROM EXECUTION TIME (NSEC)

FIGURE 5

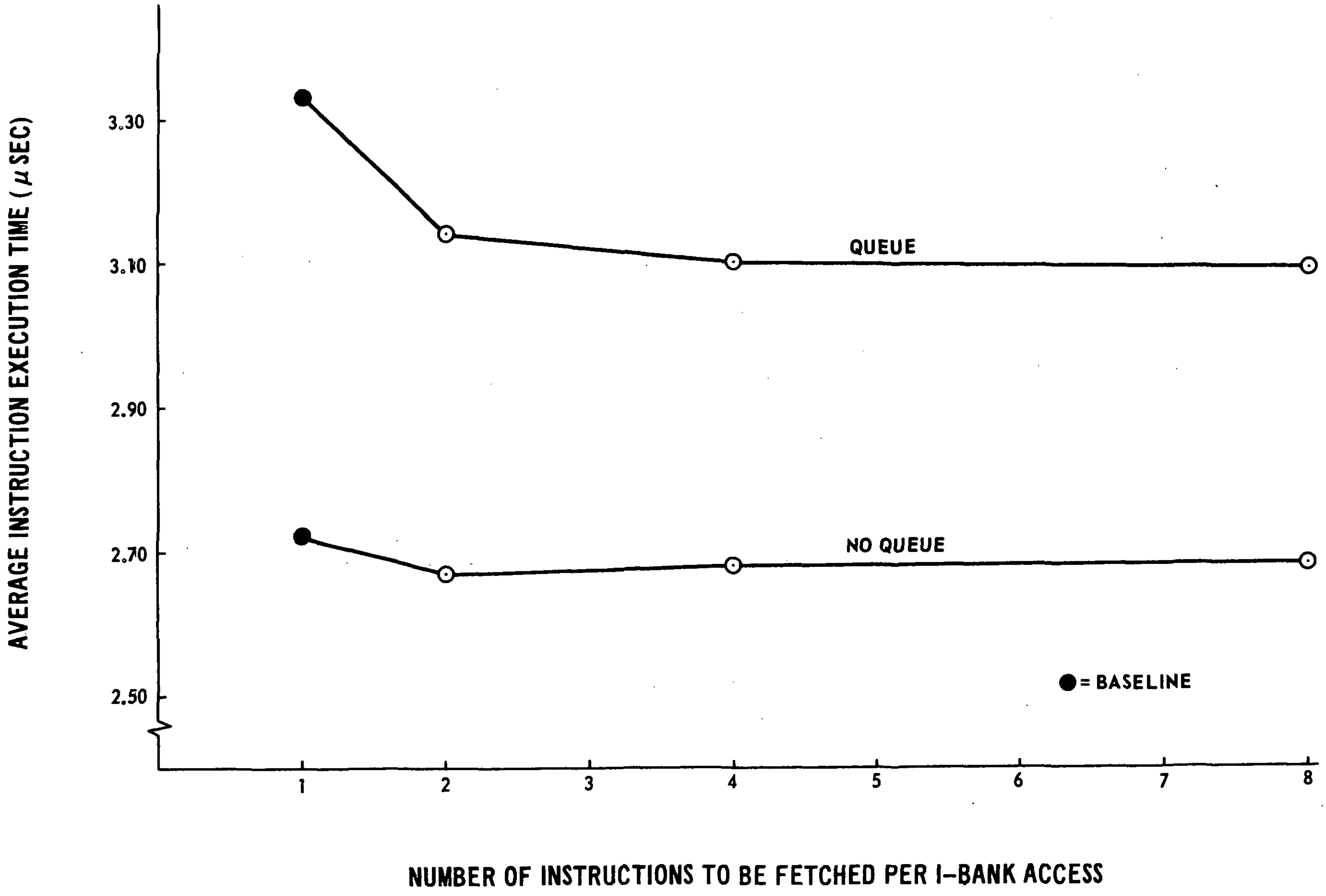

AVERAGE INSTRUCTION EXECUTION TIME (μ SEC)
3.30
3.10
2.90
2.70
2.50
QUEUE
NO QUEUE
● = BASELINE
1
2
3
4
5
6
7
8
NUMBER OF INSTRUCTIONS TO BE FETCHED PER I-BANK ACCESS

FIGURE 6

configuration. As a result of this tradeoff, it is concluded that the greatest savings, approximately 8 percent, occurred for a two-word fetch. Instruction fetches per access of greater than two instructions do not appear to be worthwhile, especially since the hardware complexity increases as the number of instructions fetched per access increases. The advantages of a multiple instruction fetch would be enhanced if there were severe memory interference problems.

D. Long/Short Instruction Execution Time and Instruction Mix

Figures 7 and 8 are included in this paper to illustrate the system dependence on two parameters whose values are difficult to determine precisely. However, if exact instruction execution times are not required, then relative speeds can be obtained from these curves, and then applied to the previous curves. Also, since both of these figures show approximately parallel "queue" and "no queue" curves, then no substantial change in memory interference has been experienced over the range of values shown.

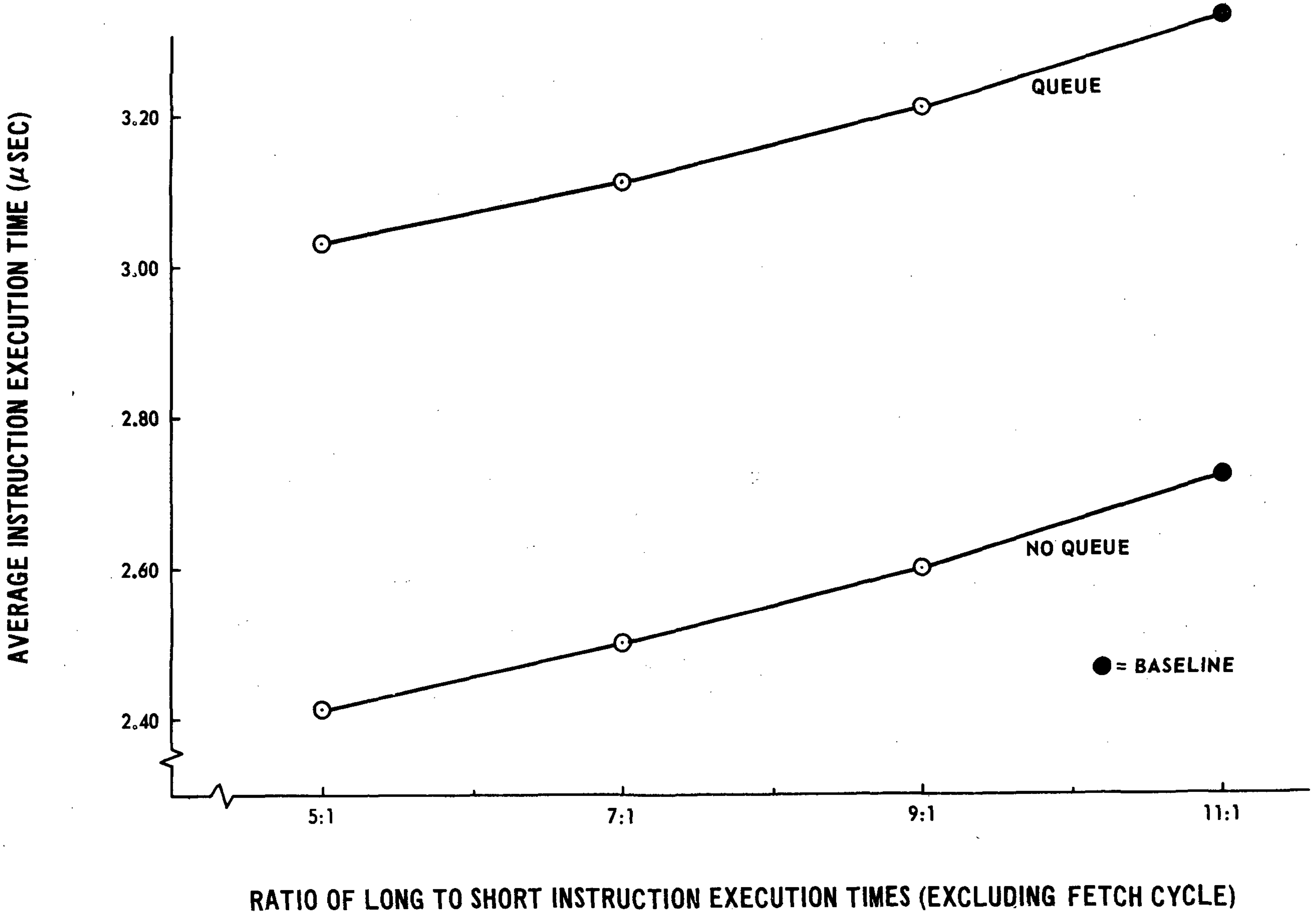

AVERAGE INSTRUCTION EXECUTION TIME (μSEC)
3.20
3.00
2.80
2.60
2.40
QUEUE
NO QUEUE
● = BASELINE
5:1
7:1
9:1
11:1
RATIO OF LONG TO SHORT INSTRUCTION EXECUTION TIMES (EXCLUDING FETCH CYCLE)

FIGURE 7

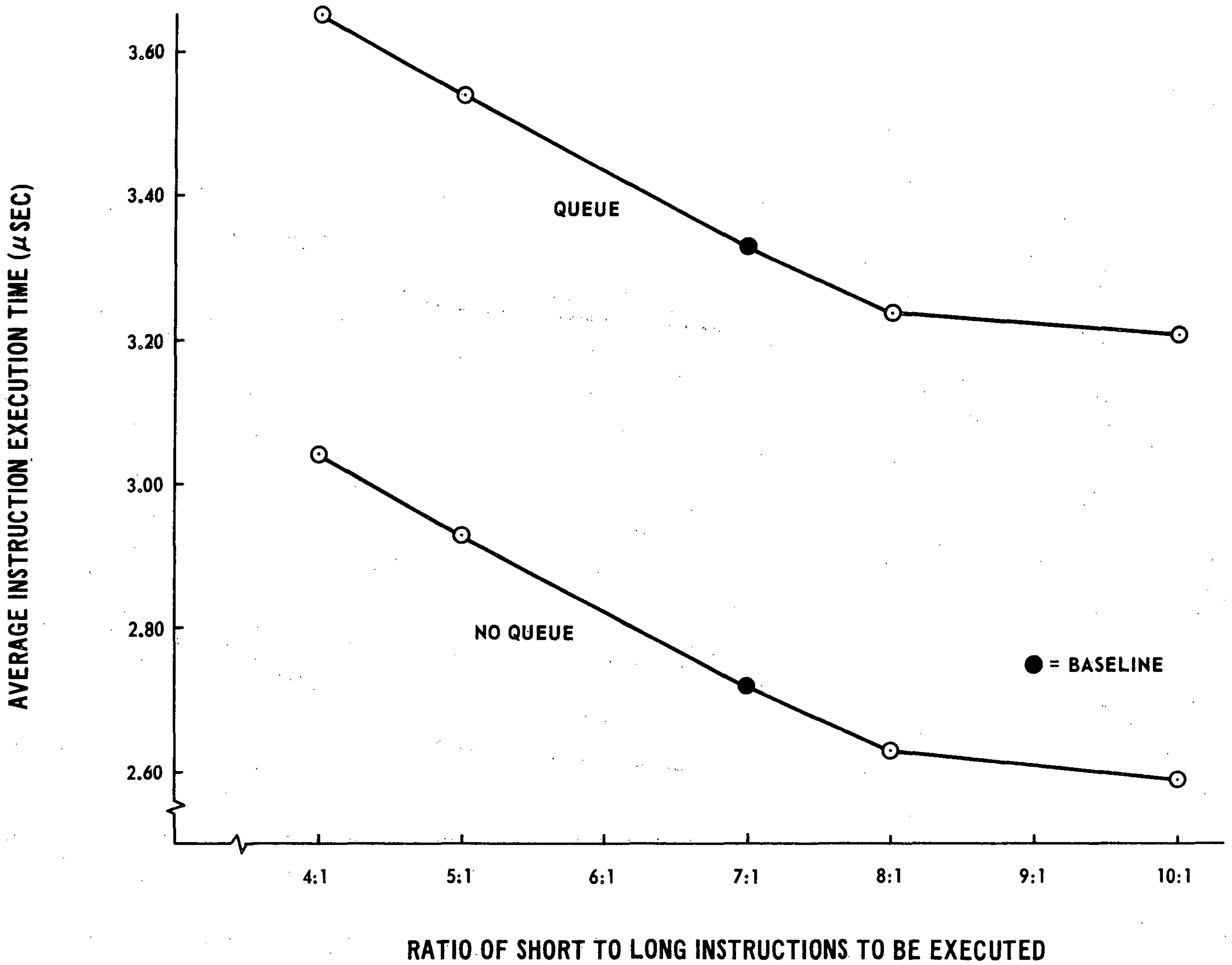

AVERAGE INSTRUCTION EXECUTION TIME (μSEC)
3.60
3.40
3.20
3.00
2.80
2.60
QUEUE
NO QUEUE
● = BASELINE
4:1
5:1
6:1
7:1
8:1
9:1
10:1
RATIO OF SHORT TO LONG INSTRUCTIONS TO BE EXECUTED

FIGURE 8

SECTION V. CONCLUSIONS

This paper has illustrated the importance of utilizing simulation as an aid in the design of a microprogrammable, modular computer. In particular, two distinct areaswhere intuition may cause the system designer some problems were made evident:

(1) Bus widths. Increasing the bus width between the CPU and memory does not necessarily mean that the average instruction execution time will decrease. Furthermore, the selection of bus widths depends greatly upon the MCROM cycle time.

(2) MROM cycle time. Increasing the MROM cycle time could result in a reduction of the average instruction execution time. Thus, one should be aware of this condition before a great deal of time and effort is spent in an attempt to reduce the MROM execution time.

As expected, multiple instruction fetches per access does reduce the average instruction execution time. However, the greatest saving is for a two-word fetch.

REFERENCES

1. Hughes Aircraft Company, "Design of a Modular Digital Computer System," DRL 4, Phase I Report, FR 72-11-450, April 15, 1972.

2. T. Williams, H. Kerner, J. Weatherbee, D. Taylor, and B. Hodges, "Optimum Spaceborne Computer System Design by Simulation", Symposium Proceedings of the XXIV AGARD Avionics Panel Technical Meeting on Automation in Manned Aerospace Systems, Oct. 1972.

3. Katz, J.H., "Simulation of a Multiprocessor Computer System," Proceedings of 1966 Spring Joint Computer Conference, pp.

4. Merikallio, R.A. and Holland, F.C., "Simulation Design of a Multiprocessing System," Fall Joint Computer Conference, 1968, pp. 1399-1409.

5. Sperry Rand Corporation, "MSFC Advanced Aerospace Computer," SP-232-0384, July 6, 1970.